WELCOME TO CS 4311
SOFTWARE ENGINEERING II
Spring 2018

Course Staff

- Instructor: Dr. Salamah Salamah
  - Email: isalamah@utep.edu
  - Office: CCS 3.0608 (phone: 747-6671)
  - Office hours: Tuesday & Thursday 1:30-2:50 pm
Course Material

- IBM SmartCloud
- Resources available
  - Course info (syllabus, contact info, etc.)
  - Lecture notes
  - Project information
  - Assignments, grading criteria, doc templates
  - Tutorials and other course-related docs

What is this course about?

- Description from the Course Catalog:
  “Methodologies, approaches, and techniques associated with design, implementation, and testing of a software system; other topics include cooperative teamwork, project management, and documentation; second semester of a two-semester capstone project in which students design and implement a real-world application specified in CS 4310.”

- Prerequisites: CS 4310 with a grade of C or better
Topics

- Configuration management
- Software design analysis techniques
- High-level software design
- Software design specification
- Software implementation
- Software validation and verification
- Software process improvement

Today’s Quote

*Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning*

Rick Cook
Learning Outcomes

Level 1: Knowledge and comprehension
- Articulate design principles, including cohesion and coupling, encapsulation, and information hiding.
- Discuss the issues related to maintenance.
- Discuss different software architectural styles, such as blackboard, event systems, layered system, and pipe and filters

Learning Outcomes (Cont.)

Level 2: Application and analysis
- Apply different diagramming techniques for an architectural design.
- Apply different textual and diagramming techniques for producing a detailed design of a system.
- Relate general strategies to identify and implement appropriate software architecture patterns according to the system being developed.
- Relate general strategies for creating a design of a system.
- Distinguish between the different levels of cohesion and coupling.
- Use software development and maintenance tools, such as software documents creation and editing tools, GUI generators, comprehension and analysis tools, supporting activities tools (configuration management tools), and verification and validation tools.
- Demonstrate skills in working as a cooperative team in order to achieve group goals.
- Describe differences between unit, integration, system, and acceptance testing.
- Apply black-box and white-box testing techniques to develop test cases for variety of test coverage.
- Engage in self-directed study to learn new techniques and tools for software design, implementation, and/or testing
Learning Outcomes (Cont.)

Level 3: Synthesis and evaluation

- Conduct a technical review of software design, implementation, and V&V.
- Create and implement a software configuration management plan.
- Create an architectural design and a detailed design for a software system.
- Construct software from a detailed design.
- Develop a test plan for a software system.
- Demonstrate an ability to orally present a software design and implementation.
- Compose software design-related documents that are grammatically correct and technically sound.
- Apply effective techniques for collaboration and problem-solving within groups.

Textbook

- **Recommended:**

- **Other resources:**
Course Structure

- Active learning complement lectures
  - In-class exercises and group work

- Project—senior capstone project
  - Refer to syllabus for expected project deliverables

- Cooperative teamwork

Course Policy - Assignments

- Reading assignments
  - To prepare for lecture and in-class work

- Project assignments
  - To apply the techniques learned to the project

- Policy
  - Late submissions will be accepted only with previous agreement (not after the fact)
Course Policy - Exams

- Two mid-term exams
- Final exam

Course Policy – Attendance

- Class attendance and participation
  - Will take attendance (cooperative learning)
  - Your final grade will be lowered by one point for each unexcused absence above three.
Course Policy - Grading

- (5 mins) Read the Grading Summary section of the syllabus (p. 3-4).
- Q: How is the final letter grade calculated?
- Q: Draw a state machine diagram for calculating the final letter grade (in your teams).

Your Turn (in your teams):

4310 Keep these for 4311
   What would you keep from CS 4310?

4310 change these
   What would you change from CS 4310?

4311 Try these
Many ways a software project fails…
Preview of CS 4311 Topics

Create a Venn Diagram that shows the set of programs that solves the problem, and of those that are feasible, maintainable, efficient

3 minutes

What CS4311 is about

- Architecture
- Design
- Implementation
- Testing
What CS4311 is about

- Architecture
- Design
- Implementation
- Testing

What are these? (3 minutes)

Outline of CS4311

- Architecture
- High-level design
- Detailed design
- Implementation
- Integration
- V&V and testing
Software Documentation

- Why?
  - Seven Principles of Sound Documentations

Why Documenting a Software Artifact?

Doing business without advertising [or designing an architecture without documenting it] is like winking at a girl in the dark. You know what you’re doing, but nobody else does.

- Steuart Henderson Britt
Why Documenting a Software Architecture? - 1

- Architecture is the blueprint for the system and the project that develops it
  - It defines the work assignments.
  - It is the primary carrier of quality attributes.
  - It is the best artifact for early analysis.
  - It is the key to post-deployment maintenance and mining.

- To be useful, this blueprint must be understood.

- To be understood, it must be communicated.

- Documentation speaks for the architect, today, tomorrow, and 20 years from now.

Why Documenting a Software Architecture? - 2

- Architecture documentation contributes to architecture design
  - Documentation enables an artifact-driven approach to software design
  - Completing the design artifact means we’ve completed the design
  - Documentation establishes the set of design decisions needed to establish/maintain the architecture
  - Making those design decisions means completing the architecture.
Documentation clarifies the line between architectural and non-architectural design decisions.

- Non-architectural design is preferred over detailed design. Architectural decisions can be quite detailed!

- Architectural design decisions affect the system's ability to deliver on its behavioral and quality goals.

- Architectural design decisions are documented in the architecture document.

Architecture documentation has three fundamental uses:

1. education, introducing people to the system: new members of the team, external analysts, the customer, or even a new architect

2. communication, as a vehicle among stakeholders and to/from the architect

3. analysis, especially for the quality attributes that the architecture design enables the system to deliver
Uses and Audience

- Architecture documentation must support these purposes:
  1. **Education**: It should be sufficiently abstract to be quickly understood by new team members.
  2. **Communication**: It should be sufficiently concrete to serve as a blueprint for construction.
  3. **Analysis**: It should provide enough information to serve as a basis for analysis.

Business Case for Architecture Documentation

- Project activities will be less costly with high quality, up-to-date documentation than they would be otherwise.
- The effort saved from architecture documentation should outweigh the cost to create it.
Principles of Sound Documentation

I have made this letter rather long only because I have not had time to make it shorter.

Blaise Pascal

Seven Principles of Sound Documentation

- These principles apply to all documentation, not just that for software architectures
  1. Write from the reader’s point of view.
  2. Avoid unnecessary repetition.
  3. Avoid ambiguity.
  4. Use a standard organization.
  5. Record your rationale.
  6. Keep documentation current, but not too current.
  7. Review documentation.
Write from the reader’s point of view

- Determine who the readers are
- Determine what readers will want to know
- Make the information concise and easy to find
  - Don’t make too many assumptions about what the readers know
- Your readers will appreciate your effort and be more likely to read your document
  - (Which will make the business case for architecture documentation)

Avoid Unnecessary Repetition

- Each kind of information should be recorded in only one place. This makes documents easier to use and change.
- Repetition often confuses the reader, especially when information is repeated in slightly different ways.
  - The reader wonders
    - “Was the difference intentional?”
    - “If so, why?”
    - “If not, which way is correct?”
- When is repetition okay?
  - Summary and overview
  - When not repeating will cause the reader to spend time flipping back through pages
    - Hyperlinks?
Avoid Ambiguity -1

Documentation is for communicating information and ideas. If the reader misunderstands because of ambiguities, the documentation has failed.

Even "simple" concepts can confuse. For example, what does the arrow above mean?
- C1 calls C2?
- Data flows from C1 to C2?
- C1 instantiates C2?
- C1 sends a message to C2?
- C1 is a subtype of C2?
- C2 is a data repository and C1 is writing data to C2?
- C1 is a repository and C2 is reading data from C1?

Avoid Ambiguity -2

Precisely defined notations/languages help avoid ambiguity.

If your documentation uses a graphical language, **always include a key!**
- It can point to the language’s formal definition.
- It can give the meaning of each symbol. (Don’t forget the lines!). If color or position is significant, indicate how.

Be sure to make the key meaningful: don't just say “element” and "relation."
- Different element and relation types should have different symbols.
Avoid Ambiguity - 3

Example of Keys

Use a Standard Organization - 1

- Establish a standard organization, make sure that your documents follow it, and make sure that readers know what it is.

- A standard organization
  - helps the reader navigate and find information
  - tells the writer what to document, where it belongs
  - helps plan the work and measure the work left to be done
  - lets the writer record information as soon as it is known, in whatever order it is discovered
  - embodies completeness rules and helps with validation.
Use a Standard Organization - 2

Corollary #1: Organize the documentation for ease of reference
- A document may be read from cover to cover only once, if at all
- A successful document will be referred to hundreds or thousands of times

Make information easy to find.

How do you do that?
- Comprehensive index
- Annotated TOC
- Reader’s guide
- Keywords, tables of figures…

Corollary #2: Don’t leave incomplete sections blank; instead, mark them “to be determined.”
- Better: “TBD by Revision 2.6”
- Better still: “TBD by 14 November 2011”

Why?

Corollary #3: If a section doesn’t apply, don’t leave it blank or delete it;
- mark it “not applicable.”
- Better: “not applicable because…

Why?
Record Your Rationale

- Why did you make certain design decisions?
- Next week or next year, how will you remember? How will the next architect know?
- Recording your rationale requires discipline but can save enormous time in the long run.
- Record significant rejected alternatives as well.
  - helps avoid wasting time on the same dead ends in the future
  - might explain when a dead end is no longer dead

Keep Documentation Current, But Not Too Current.

- This rule applies throughout the entire life cycle of the system
- Documentation that is incomplete or out of date
  - does not reflect the truth
  - disobeys its own rules about form and internal consistency
  - is not likely to be used
- Documentation that is kept current
  - can provide quick and efficient answers to questions about the software
  - is more likely to be used
Help instill a *documentation-based culture* in your organization by letting documents answer questions.

- The architect's first answer should be “Here is where you can find that information in the documentation.”
- If the information is missing, update the document.
- Make sure the next release contains the information.
- Send the message that the *documentation* is the preferred, authoritative source for information.

Contrast that to the architect who happily answers questions every time the phone rings.

Don’t keep it *too* current:

- During the design process, decisions are considered and reconsidered frequently.
- Releasing too often will result in unnecessary expense and frustration among the readers.
- Determine points in the development process when up-to-date documentation will be released.
- Follow a release strategy or *rhythm* that makes sense for your project.
Review Documentation

- Only the intended users of a document can tell you if it
  - contains the right information
  - presents the information in a useful way
  - satisfies their needs

- Plan to review your documents with representatives of the stakeholders for whom it was created.

Summary

- CS 4311 should be a fun class
  - Lots of work, but organization and timeliness are key

- Artifact driven software development does not mean documentation for the sake of documentation

- You should follow sound documentation principles for the reward for your documentation to outweigh the effort and time it takes to create these documents.